Improved Binary Artificial Fish Swarm Algorithm for the 0–1 Multidimensional Knapsack Problems

Md. Abul Kalam Azad[a,*], Ana Maria A.C. Rocha[a,b], Edite M.G.P. Fernandes[a]

[a] *Algoritmi R&D Centre*
[b] *Department of Production and Systems*
School of Engineering, University of Minho, 4710-057 Braga, Portugal

Abstract

The 0–1 multidimensional knapsack problem (MKP) arises in many fields of optimization and is NP-hard. Several exact as well as heuristic methods exist. Recently, an artificial fish swarm algorithm has been developed in continuous global optimization. The algorithm uses a population of points in space to represent the position of fish in the school. In this paper, a binary version of the artificial fish swarm algorithm is proposed for solving the 0–1 MKP. In the proposed method, a point is represented by a binary string of 0/1 bits. Each bit of a trial point is generated by copying the corresponding bit from the current point or from some other specified point, with equal probability. Occasionally, some randomly chosen bits of a selected point are changed from 0 to 1, or 1 to 0, with an user defined probability. The infeasible solutions are made feasible by a decoding algorithm. A simple heuristic add_item is implemented to each feasible point aiming to improve the quality of that solution. A periodic reinitialization of the population greatly improves the quality of the solutions obtained by the algorithm. The proposed method is tested on a set of benchmark instances and a comparison with other methods available in literature is shown. The comparison shows that the proposed method gives a competitive performance when solving this kind of problems.

Keywords: 0–1 knapsack problem, multidimensional knapsack, artificial fish swarm, decoding algorithm

*Corresponding author
Email addresses: akazad@dps.uminho.pt (Md. Abul Kalam Azad), arocha@dps.uminho.pt (Ana Maria A.C. Rocha), emgpf@dps.uminho.pt (Edite M.G.P. Fernandes)

1. Introduction

The 0–1 multidimensional knapsack problem (MKP) is a NP-hard combinatorial optimization problem that arises in many practical problems, such as capital budgeting and project selection problem [1, 2], allocating processors and databases in a distributed computer system [3], project selection, cargo loading and so on [4]. The 0–1 MKP is formulated as follows:

$$\begin{aligned} \text{maximize} \quad & z(\boldsymbol{x}) \equiv \boldsymbol{c}\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & x_j \in \{0,1\}, \quad j = 1, 2, \ldots, n, \end{aligned} \tag{1}$$

where $\boldsymbol{c} = (c_1, c_2, \ldots, c_n)$ is an n-dimensional row vector of profits, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ is an n-dimensional column vector of 0–1 decision variables, $\boldsymbol{A} = [a_{k,j}]$, $k = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ is an $m \times n$ coefficient matrix of resources and $\boldsymbol{b} = (b_1, b_2, \ldots, b_m)^T$ is an m-dimensional column vector of resource capacities. It should be noted here that, in a 0–1 multidimensional knapsack problem, each element of $\boldsymbol{c}$, $\boldsymbol{A}$ and $\boldsymbol{b}$ is assumed to be nonnegative. The goal of the 0–1 MKP is to find a subset of n items that yields maximum profit z without exceeding resource capacities $\boldsymbol{b}$. There are many knapsack-like problems. The knapsack family includes the 0–1 knapsack problem (KP), where there is just one single constraint ($m = 1$). Effective approximate algorithms have been developed for obtaining its near optimal solutions. If a bounded amount of each item type is available, then the bounded knapsack problem arises. On the other hand, the unbounded knapsack problem is a generalization of the bounded knapsack problem where an unlimited number of each item type is available. Another generalization of the 0–1 knapsack problem is the multiple-choice knapsack problem where exactly one item from each of several classes of items is chosen such that the profit z is to be maximized. If the profits equal the resources $c_j = a_j$, $j = 1, 2, \ldots, n$, then one has the subset-sum problem. The name comes from the fact that it can be seen as the problem of choosing a subset of $c_1, c_2, \ldots, c_n$ such that its sum is as large as possible without exceeding b. The multiple knapsack problem appears when some of the n items are chosen to pack in m knapsacks of (maybe) different capacities such that the profit z is maximized. The most general form of the knapsack problem is the multidimensional knapsack problem where all coefficients $c_j, a_{k,j}, b_k$, $k = 1, 2, \ldots, m, j = 1, 2, \ldots, n$ and variables $x_j, j = 1, 2, \ldots, n$ are nonnegative integers, which turns out to be a general integer programming problem [5].

In the last decades several exact as well as heuristic methods have been proposed to solve the MKP. Exact methods include dynamic programming methods [6, 7], branch-and-bound algorithms [3, 4, 8], the Fourier-Motzkin elimination based enumeration algorithms [9], asymptotic analysis method [10], statistical analysis

method [11], linked LP-relaxations, disjunctive cuts and implicit enumeration [12], generalized fuzzy approach [13], core concept based on LP-relaxation [14] and so on. Pisinger [5] has proposed several exact algorithms for solving knapsack problems in his doctoral thesis. A list of some heuristic methods for solving the MKP follows. Drexl [15] proposed a simulated annealing based on the add-interchange-drop technique for handling the constraints. Hanafi and Fréville [16] proposed a tabu search approach for the 0–1 MKP using the surrogate constraints information. Vasquez and Vimont in [17] presented a hybrid method that combines the linear programming with an efficient tabu search. Chu and Beasley [18] proposed the most successful genetic algorithm (GA) for solving the 0–1 MKP. The authors present the drop-add repair operator based on the pseudo-utility ratios in order to make the solutions feasible. Sakawa and Kato [19] introduced a genetic algorithm with double strings (GADS) based on a decoding algorithm. In this decoding algorithm, the items that make the solution infeasible are dropped from the solution by checking all the constraints. Djannaty and Doosdar investigated in [20] a hybrid genetic algorithm that uses a penalty function. A binary ant colony optimization algorithm based on the drop-add repair operator [18] is provided in [21]. Zou et al. [22] have recently developed a novel global harmony search algorithm based on a penalty function for a KP. Some other heuristics are available in the literature [23, 24, 25, 26, 27]. An interesting review of different solution methods for solving the 0–1 MKP is found in [28]. The focus of the paper is on the theoretical properties and contains an overview of approximate and exact solution methods.

The artificial fish swarm algorithm (AFSA) that simulates the behavior of a fish school inside water was recently designed and applied in an engineering context [29, 30, 31, 32]. Fishes desire to stay close to the school to protect themselves from predators and to look for food, and to avoid collisions within the group. The main fish school behavior are the following: random, chasing, swarming, searching and leaping.

The artificial fish is a fictitious entity of a true fish. When applied to an optimization problem, a 'fish' within the school is represented by a point, also known as a candidate solution, and the school is the so-called population, or set of points or solutions. Inspired by fish school behavior, researchers have developed numerical algorithms aiming to converge to a global optimal solution of the optimization problem, in an efficient manner. The environment in which the artificial fish moves, searching for the optimum, is the feasible search space of the problem.

A novel fish swarm heuristic which gives priority to the chasing behavior in detriment of the swarming one, for box constrained global optimization problems, was recently presented in [33]. Rocha et al. [34] developed an augmented Lagrangian fish

swarm based method for globally solving a nonlinear general constrained problem. A state-of-the-art regarding hybridizations and applications of the AFSA has just appeared in [35].

Binary versions of the most popular stochastic algorithms are common for solving discrete binary optimization problems [36, 37], namely 0–1 MKP [38, 39, 40, 41, 42]. Based on AFSA for continuous global optimization, in this paper, we propose an improved binary version of the artificial fish swarm algorithm (IbAFSA) for solving the 0–1 MKP (1). A preliminary binary version of the artificial fish swarm algorithm (bAFSA) has been presented in [43]. The algorithm was tested on a small set of problems. For the sake of simplicity, while describing the proposed binary AFSA we will use the words 'point' to represent the position of a fish in the school, and 'population' to denote the fish school.

In the present study, all points in the population are randomly initialized, each represented by a binary 0/1 string of length n. The procedure that checks which points are in the vicinity of each individual point, the so-called 'visual scope', is carried out using the Hamming distance. When chasing, searching or swarming behavior are selected, the proposed IbAFSA generates each bit of the trial point by copying the corresponding bit from the current point or from a second point, with equal probability. In chasing, the second point is the best point inside the 'visual scope', and in searching, that point is randomly selected from the 'visual scope'. For the swarming behavior, the second point is the central point that is computed based on ideas presented in [44]. We remark that in the previous work [43], when swarming was implemented, a bit of the current point was randomly selected and changed from 0 to 1 or vice versa to create the trial point. Furthermore, the infeasible solutions are made feasible using an adapted version of the decoding algorithm presented in [19]. Along with the decoding algorithm, an add_item operation is also implemented to each feasible solution aiming to increase the profit throughout the addition of more items in the knapsack. To improve the quality of the solutions obtained by the algorithm, the population is periodically reinitialized.

Thus, the novel contributions of the presented IbAFSA, when compared with bAFSA [43], are: i) the computation of a central point inside the 'visual scope' to define a point that is closest to all the other points in the 'visual scope', for the swarming behavior; ii) the implementation of a different strategy to generate the trial point using the current and the central point, in swarming; iii) the implementation of an add_item operation to each feasible point; iv) the reinitialization of the population periodically, although keeping the best point of the population. The performance of the proposed IbAFSA is tested on a benchmark set of 0–1 MKP test instances. Although the proposal is very simple and easy-to-implement, the comparisons carried

out until now show that the algorithm is a competitive alternative to other heuristic methods from the literature.

A crucial motivation to assess the performance of IbAFSA on the 0–1 MKP is that several test problem instances together with their known optimal solution are available in the literature.

The organization of this paper is as follows. We briefly describe the artificial fish swarm algorithm in Section 2. In Section 3 the proposed improved binary artificial fish swarm algorithm is outlined. Section 4 describes the experimental results and finally we draw the conclusions of this study in Section 5.

2. Artificial Fish Swarm Algorithm

In this section, we give a brief description of AFSA proposed in [33] for box constrained global optimization problems of type minimize$_{\boldsymbol{x}\in\Omega}$ $f(\boldsymbol{x})$. Here $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function that is to be minimized and $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n : l_j \leq x_j \leq u_j, j = 1, 2, \ldots, n\}$ is the search space. l_j and u_j are the lower and upper bounds of x_j, respectively, and n is the number of variables of the optimization problem.

AFSA works with a population of N points $\boldsymbol{x}^i, i = 1, 2, \ldots, N$ to identify promising regions looking for a global solution [31]. $\boldsymbol{x}^i$ is a floating-point encoding that covers the entire search space Ω. The crucial issue of AFSA is the 'visual scope' of each point $\boldsymbol{x}^i$. This represents a closed neighborhood of $\boldsymbol{x}^i$ with a radius equal to a positive quantity ν defined by

$$\nu = \delta \max_{j\in\{1,2,\ldots,n\}} (u_j - l_j)$$

where $\delta \in (0, 1)$ is a positive visual parameter. This parameter may be reduced along the iterative process. Let $\boldsymbol{I}^i$ be the set of indices of the points inside the 'visual scope' of point $\boldsymbol{x}^i$, where $i \notin \boldsymbol{I}^i$ and $\boldsymbol{I}^i \subset \{1, 2, \ldots, N\}$, and let np^i be the number of points in its 'visual scope'. Depending on the relative positions of the points in the population, three possible situations may occur:

a) when $np^i = 0$, the 'visual scope' is empty, and the point $\boldsymbol{x}^i$, with no other points in its neighborhood, moves randomly looking for a better region;

b) when the 'visual scope' is not crowded, the point $\boldsymbol{x}^i$ is able either to chase moving towards the best point inside the 'visual scope', or, if this best point does not improve the objective function value corresponding to $\boldsymbol{x}^i$, to swarm moving towards the central point of the 'visual scope';

c) when the 'visual scope' is crowded, the point $\boldsymbol{x}^i$ has some difficulty in following any particular point, and searches for a better region by choosing randomly another point (from the 'visual scope') and moving towards it.

The condition that decides when the 'visual scope' of $\boldsymbol{x}^i$ is not crowded is

$$C_f \equiv \frac{np^i}{N} \leq \theta, \tag{2}$$

where C_f is the crowding factor and $\theta \in (0, 1)$ is the crowd parameter. In this situation, the point $\boldsymbol{x}^i$ has the ability to swarm or to chase. The swarming behavior is characterized by a movement towards the central point inside the 'visual scope' of $\boldsymbol{x}^i$ defined by

$$\bar{\boldsymbol{x}} = \frac{\sum_{l \in \boldsymbol{I}^i} \boldsymbol{x}^l}{np^i}.$$

We refer the reader to [31, 32, 33, 34] for details.

3. Improved Binary Artificial Fish Swarm Algorithm

In this section we will present the proposed IbAFSA to solve the 0–1 multidimensional knapsack problem (1). The outline of the algorithm is described in the following.

3.1. Initialization (coding)

The first step to design the IbAFSA for solving the 0–1 MKP is to devise a suitable representation scheme of a point/solution from the population. Since we consider the 0–1 knapsack problem, N solutions, $\boldsymbol{x}^i, i = 1, 2, \ldots, N$ are randomly initialized, each represented by a binary 0/1 string of length n [43, 45, 46]. We remark that the maximum population size N of binary 0/1 strings of length n is 2^n.

3.2. Generating trial points in IbAFSA

In IbAFSA the Hamming distance, H_d, is used to identify the points inside the 'visual scope' of point $\boldsymbol{x}^i$. The Hamming distance between two bit sequences of equal length is the number of positions at which the corresponding bits are different. After calculating the Hamming distance between all pair of points from the population, the np^i points inside the 'visual scope' of $\boldsymbol{x}^i$ are identified as the points $\boldsymbol{x}^j$ that satisfy the condition $H_d(\boldsymbol{x}^i, \boldsymbol{x}^j) \leq \nu$, for $j \in \{1, \ldots, N\}$, $j \neq i$, where

$$\nu = \delta \times n, \tag{3}$$

$\delta \in (0, 1)$ and n represents the maximum Hamming distance between two binary points. After computing np^i, the crowding factor C_f of $\boldsymbol{x}^i$ is calculated using (2).

Depending on the value of C_f, the 'visual scope' can be empty, not crowded or crowded. In IbAFSA, the behavior that generate the trial points are outlined as follows.

3.2.1. Chasing behavior

If the 'visual scope' of $\boldsymbol{x}^i$ is not crowded and the point that has the best objective function value inside the 'visual scope', denoted by $\boldsymbol{x}^{\text{best}}$ (best $\in \boldsymbol{I}^i$), satisfies $z(\boldsymbol{x}^{\text{best}}) > z(\boldsymbol{x}^i)$, the chasing behavior is to be implemented. In chasing, each bit of the trial point, $\boldsymbol{y}^i$, is generated by copying the corresponding bit from $\boldsymbol{x}^i$ or from $\boldsymbol{x}^{\text{best}}$ with equal probability. This operation is similar to the uniform crossover present in genetic/evolutionary algorithms.

3.2.2. Swarming behavior

When the 'visual scope' is not crowded and $z(\boldsymbol{x}^{\text{best}}) \leq z(\boldsymbol{x}^i)$ (chasing is not possible), then if $z(\bar{\boldsymbol{x}}) > z(\boldsymbol{x}^i)$, where $\bar{\boldsymbol{x}}$ is the central point inside the 'visual scope' of the point $\boldsymbol{x}^i$, the swarming behavior is to be implemented. The central $\bar{\boldsymbol{x}}$ is the point closest to all the other points in the 'visual scope', in the sense that the average Hamming distance to all other points in the 'visual scope' is minimal. Since in IbAFSA, the points are represented by binary 0/1 strings, each bit of $\bar{\boldsymbol{x}}$ takes the majority of the corresponding bits of the other points in the 'visual scope', and is randomly defined in case of tie. We refer to [44] for details. The pseudocode to compute the central point is shown in Algorithm 1. In swarming, each bit of the trial $\boldsymbol{y}^i$ is created by copying the corresponding bit from $\boldsymbol{x}^i$ or from $\bar{\boldsymbol{x}}$ with equal probability.

Algorithm 1 Central point

Require: Set $\boldsymbol{I}^i$ and the np^i points inside the 'visual scope' of $\boldsymbol{x}^i$

for $j = 1$ to n **do**

  Compute $\bar{x}_j = \dfrac{\sum_{l \in \boldsymbol{I}^i} x_j^l}{np^i}$

  if $\bar{x}_j = 0.5$ **then**

    Set $\bar{x}_j :=$ Random Integer$\{0, 1\}$

  else

    Set $\bar{x}_j :=$ Round$(\bar{x}_j)$

  end if

end for

return Central point $\bar{\boldsymbol{x}}$

3.2.3. Searching behavior

The searching behavior is tried in the following situations:

a) when the 'visual scope' is not crowded and neither $\boldsymbol{x}^{\text{best}}$ nor $\bar{\boldsymbol{x}}$ improves in objective function value;

b) when the 'visual scope' is crowded.

Here, a point $\boldsymbol{x}^{\text{rand}}$ (rand $\in \boldsymbol{I}^i$) inside the 'visual scope' of $\boldsymbol{x}^i$ is randomly selected and the searching behavior is to be implemented if $z(\boldsymbol{x}^{\text{rand}}) > z(\boldsymbol{x}^i)$. Otherwise, a random behavior is implemented. In searching, each bit of $\boldsymbol{y}^i$ is created by copying the corresponding bit from $\boldsymbol{x}^i$ or $\boldsymbol{x}^{\text{rand}}$ with equal probability.

3.2.4. Random behavior

When the 'visual scope' of $\boldsymbol{x}^i$ is empty or the other behavior were not performed, the point $\boldsymbol{x}^i$ performs the random behavior. In this case, the trial point $\boldsymbol{y}^i$ is created by randomly setting a binary string of 0/1 bits of length n.

3.3. Constraints handling

The widely used approach to deal with constraints is based on penalty functions where a penalty term is added to the objective function in order to penalize the constraint violation. The penalty function method can be applied to any type of constraints, but the performance of penalty-type method is not always satisfactory due to the choice of an appropriate penalty parameter. Although several ideas have been proposed about designing efficient penalty functions and tuning penalty parameters [20, 22], other alternative constraint handling techniques have emerged in the last decades.

There are a number of standard ways of dealing with constraints and infeasible solutions in binary represented population-based methods. In IbAFSA, the decoding algorithm proposed by Sakawa and Kato [19] to make infeasible solutions feasible is used. Although GADS and IbAFSA use different point representations, we modify the decoding algorithm so that it can decode points in a population in the same way as in [19]. The advantage of this algorithm is that decoding a point $\boldsymbol{x}^i$ starts from any index and randomly continues to select an index until the maximum length of string n is reached to make the point $\boldsymbol{x}^i$ feasible, aiming to obtain promising solution (and hopefully optimal). At first, a set $\boldsymbol{J} = \{J_1, J_2, \ldots, J_n\}$ is defined with n randomly generated indices. Then the decoding algorithm is performed on $\boldsymbol{x}^i$ using the set $\boldsymbol{J}$ to make it feasible. This means that, using the sequence $\boldsymbol{J}$, and one item/bit at a time all constraints are checked for capacity satisfaction, using the corresponding

column of the coefficient matrix of the resources. If all constraints are satisfied, the bit 1 is maintained and the item is stored in the knapsack. Otherwise, the bit is changed to 0. See Algorithm 2. Another decoding algorithm which starts from the

Algorithm 2 Decoding algorithm used in IbAFSA

Require: Point $\boldsymbol{x}^i$ and the set $\boldsymbol{J} = \{J_1, J_2, \ldots, J_n\}$
Set $\text{sum}_k := 0$, for $k = 1, 2, \ldots, m$
for $j = 1$ to n **do**
  if $x^i_{J_j} = 1$ **then**
    Set flag := 1
    for $k = 1$ to m **do**
      if $\text{sum}_k + a_{k,J_j} > b_k$ **then**
        Set flag := 0
        break
      end if
    end for
    if flag $= 1$ **then**
      for $k = 1$ to m **do**
        Set $\text{sum}_k := \text{sum}_k + a_{k,J_j}$
      end for
    else
      Set $x^i_{J_j} := 0$
    end if
  end if
end for
return Feasible point $\boldsymbol{x}^i$

first index and sequentially continues can be applied but the obtained solution may not be optimal.

After the decoding algorithm, a simple greedy-like heuristic called add_item (Algorithm 3) is implemented to each feasible solution aiming to improve that point without violating any constraint. When solving the single knapsack problem, this heuristic operation uses the information of the *pseudo-utility* ratios, δ_j, which are defined as the ratios of the objective function coefficients (c_j's) to the coefficients of the single constraint (a_j's). The greater the ratio, the higher the chance that the corresponding variable will be equal to one in the solution [18]. In the generalization of this add_item heuristic for the 0-1 MKP, the *pseudo-utility* ratios of every item in every constraint are calculated, and only the lowest value for each item is considered (i.e., $\delta_j = \min\{(c_j b_k)/a_{k,j}\}$ $j = 1, \ldots, n, k = 1, \ldots, m$). Then δ_j are sorted in decreasing order and a set $\boldsymbol{J} = \{J_1, J_2, \ldots, J_n\}$ is defined with the indices of the δ_j in decreasing order. One item is added each time in the knapsack if it satisfies all the constraints following the sequence of indices in the set $\boldsymbol{J}$. This procedure is

continued until the entire sequence of indices has been used.

Algorithm 3 Add_item algorithm used in IbAFSA

Require: Feasible point $\boldsymbol{x}^i$ and set $\boldsymbol{J} = \{J_1, J_2, \ldots, J_n\}$
Compute $\text{sum}_k = \sum_{j=1}^{n} a_{k,j} x_j^i$, for $k = 1, 2, \ldots, m$
for $j = 1$ to n **do**
  if $x_{J_j}^i = 0$ **then**
    Set flag := 1
    for $k = 1$ to m **do**
      if $\text{sum}_k + a_{k,J_j} > b_k$ **then**
        Set flag := 0
        break
      end if
    end for
    if flag = 1 **then**
      Set $x_{J_j}^i := 1$
      for $k = 1$ to m **do**
        Set $\text{sum}_k := \text{sum}_k + a_{k,J_j}$
      end for
    end if
  end if
end for
return Improved feasible point $\boldsymbol{x}^i$

3.4. Selection of a new population

Among the trial points $\boldsymbol{y}^{i,t}$ and the current points $\boldsymbol{x}^{i,t}$, $i = 1, 2, \ldots, N$, at iteration t, in order to decide whether or not they should become members of the population in the next iteration, $t+1$, the trial point is compared to the current point using the following greedy criterion:

$$\boldsymbol{x}^{i,t+1} = \begin{cases} \boldsymbol{y}^{i,t} & \text{if } z(\boldsymbol{y}^{i,t}) \geq z(\boldsymbol{x}^{i,t}) \\ \boldsymbol{x}^{i,t} & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, N. \tag{4}$$

3.5. Leaping behavior

When the best objective function value in the population does not change for a certain number of iterations, the algorithm may have stagnated. The other points of the population will eventually converge to that objective function value. To be able to escape from this region and to try to converge to the optimal solution, the algorithm performs the leaping behavior, at every L iterations. In the leaping, a point $\boldsymbol{x}^{\text{rand}}$ (rand $\in \{1, 2, \ldots, N\}$) is randomly selected from the current population and some randomly selected bits of the point are changed from 0 to 1 or vice versa

with probability p_m. The value $p_m = 0.01$ is widely used in binary represented methods. The described operation is similar to a mutation with probability p_m of genetic/evolutionary algorithms.

Afterwards, decoding and the add_item heuristic are implemented, and the new point replaces the point $\boldsymbol{x}^{\text{rand}}$.

3.6. Termination conditions

Let $T_{\max}$ be the maximum number of iterations. Let $z_{\max}$ be the maximum objective function value attained at iteration t and z_{opt} be the known optimal value available in the literature. The proposed IbAFSA terminates if one of the conditions

$$t > T_{\max} \text{ or } |z_{\max} - z_{\text{opt}}| \leq \epsilon \tag{5}$$

holds, where ϵ is a small positive tolerance. This condition enables the algorithm to terminate when the best known solution with a tolerance ϵ is reached; otherwise, it continues execution until $T_{\max}$ is exceeded. However, if the optimal value of the given problem is unknown, the algorithm may use other termination conditions.

3.7. Reinitialization of the population

Past experiments with bAFSA [43] have shown that, from a certain iteration on, all the individual points in a population converge to a non-optimal point, even after the leaping behavior has been performed. To diversify the search, we propose to reinitialize the population randomly, every R iterations, keeping the best solution found so far. In practical terms, this technique has greatly improved the quality of the solutions and increased the consistency of the proposed improved binary version of AFSA.

3.8. The algorithm

The pseudocode of the herein proposed improved binary version of AFSA for solving the 0–1 multidimensional knapsack problem (1) is shown in Algorithm 4.

3.9. Time complexity of one iteration of IbAFSA

The algorithm time complexity is usually measured using $\mathcal{O}$ notation and shows how the amount of time needed to complete the (operations in the) algorithm varies as the size of the input data m and n increase. The time complexity of one iteration, for the worst-case scenario of the Algorithm 4, is analyzed assuming that we have a population of N points, each point is represented by an n-vector and the problem has m constraints. The computation for each iteration is as follows.

Algorithm 4 IbAFSA

Require: $T_{\max}$ and z_{opt} and other values of parameters
Set $t := 1$. Initialize population $\boldsymbol{x}^{i,1}, i = 1, 2, \ldots, N$
Perform decoding and add_item, evaluate the population and identify $\boldsymbol{x}^{\max}$ and $z_{\max}$
while 'termination conditions are not met' **do**
  if MOD(t,R)=0 **then**
    Reinitialize population $\boldsymbol{x}^{i,t}, i = 1, 2, \ldots, N-1$
    Perform decoding and add_item, evaluate population and identify $\boldsymbol{x}^{\max}$ and $z_{\max}$
  end if
  for all $\boldsymbol{x}^{i,t}$ **do**
    Compute 'visual scope' and 'crowding factor'
    if 'visual scope' is empty **then**
      Perform random behavior to create trial point $\boldsymbol{y}^{i,t}$
    else if 'visual scope' is not crowded **then**
      if $z(\boldsymbol{x}^{\text{best}}) > z(\boldsymbol{x}^{i,t})$ **then**
        Perform chasing behavior to create trial point $\boldsymbol{y}^{i,t}$
      else if $z(\bar{\boldsymbol{x}}) > z(\boldsymbol{x}^{i,t})$ **then**
        Perform swarming behavior to create trial point $\boldsymbol{y}^{i,t}$
      else if $z(\boldsymbol{x}^{\text{rand}}) > z(\boldsymbol{x}^{i,t})$ **then**
        Perform searching behavior to create trial point $\boldsymbol{y}^{i,t}$
      else
        Perform random behavior to create trial point $\boldsymbol{y}^{i,t}$
      end if
    else if 'visual scope' is crowded **then**
      if $z(\boldsymbol{x}^{\text{rand}}) > z(\boldsymbol{x}^{i,t})$ **then**
        Perform searching behavior to create trial point $\boldsymbol{y}^{i,t}$
      else
        Perform random behavior to create trial point $\boldsymbol{y}^{i,t}$
      end if
    end if
  end for
  Perform decoding and add_item to get $\boldsymbol{y}^{i,t}, i = 1, 2, \ldots, N$ and evaluate them
  Select new population $\boldsymbol{x}^{i,t+1}, i = 1, 2, \ldots, N$
  if MOD(t,L)=0 **then**
    Perform leaping behavior, decoding, add_item and evaluate
  end if
  Identify $\boldsymbol{x}^{\max}$ and $z_{\max}$
  Set $t := t + 1$
end while
return $\boldsymbol{x}^{\max}$ and $z_{\max}$

1 Step 1, the initialization, takes Nn operations;
2 Step 2, decoding and add_item take Nmn and evaluating the population takes N;
3 the total time is $N(mn + 1)$;

Step 4 – Step 7 take $(N-1)n$ (for reinitialization of $N-1$ points), $(N-1)mn$ for decoding and add_item, and $N-1$ for evaluation, i.e., the total is $(N-1)(n+mn+1)$;
Step 8 – Step 29: to compute the 'visual scope' of each point and to check which points are in its vicinity take n^2; to generate the trial point takes n; thus, when all N points are considered, the total time is Nn^2+Nn;
Step 30 takes Nmn;
Step 31 takes N;
Step 32 – Step 34 take mn;

Adding everything up $Nn+N(mn+1)+(N-1)(n+mn+1)+Nn(n+1)+Nmn+N+mn$ gives a time of $N(3mn+3n+3+n^2)$. Considering that N is a constant, the complexity is of $\mathcal{O}(n^2)$ for fixed m, $\mathcal{O}(m)$ for fixed n and $\mathcal{O}(mn+n^2)$ for variable m and n.

4. Experimental Results

We code IbAFSA in C and compile with Microsoft Visual Studio 10.0 compiler in a PC having 2.5 GHz Intel Core 2 Duo processor and 4 GB RAM. We set $N=100$, $\delta=0.5$, $\theta=0.8$, $p_m=0.01$ and $\epsilon=10^{-4}$. In order to perform the leaping behavior, we set $L=\max(25,n)$. After several experiments, we set the parameter R for the reinitialization of the population to 100. We consider six benchmark sets of 0–1 MKP with a total of 55 instances from OR-library[1]. These problems are widely used for the measurement of effectiveness of an algorithm in the optimization community. The number of variables, n, in the instances varies from six to 105, and m (number of constraints) varies from two to 30. Table 1 lists the values of n and m of the instances for each problem set. Since they are benchmark instances, the optimal solution, z_{opt}, is known and the termination condition described in (5) can be used to terminate the algorithm. For these instances, we set $T_{\max}=1000$ if $n\leq 50$; otherwise 2000.

First, we compare IbAFSA with CPLEX MIP solver, GA [18], bAFSA [43] and GADS [19]. We run CPLEX MIP solver in our computer to solve the instances and report the obtained results. We use the data of GA available in the corresponding literature [18]. We note that GA uses a different termination condition and performs just a single run for each instance. We also code GADS in C and run with the recommended parameters [19]. In GADS, partially matched crossover, bit flip mutation and inversion are used. The crossover, mutation and inversion probabilities are set to 0.9, 0.1 and 0.03 respectively. We use the same termination conditions (5) for

[1] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`

Table 1: Values of (n, m) of each instance in the sets

Problem set	Number of instances	(n, m)
HP	2	(28, 4), (35, 4)
PB	6	(27, 4), (34, 4), (29, 2), (20, 10), (40, 30), (37, 30)
PT	7	(6, 10), (10, 10), (15, 10), (20, 10), (28, 10), (39, 5), (50, 5)
SENTO	2	(60, 30), (60, 30)
WEING	8	(28, 2), (28, 2), (28, 2), (28, 2), (28, 2), (28, 2), (105, 2), (105, 2)
WEISH	30	(30, 5), (30, 5), (30, 5), (30, 5), (30, 5), (40, 5), (40, 5), (40, 5), (40, 5), (50, 5), (50, 5), (50, 5), (50, 5), (60, 5), (60, 5), (60, 5), (60, 5), (70, 5), (70, 5), (70, 5), (70, 5), (80, 5), (80, 5), (80, 5), (80, 5), (90, 5), (90, 5), (90, 5), (90, 5), (90, 5)

GADS and run the program 30 times to report the results. We also run bAFSA and IbAFSA 30 times for each instance. In this comparison, GA, bAFSA, GADS and IbAFSA had the same value of N. The comparative results are shown in Table 2.

The table reports the average number of iterations, 'AIT', and the average computational time (in seconds), 'AT', considering all the 30 runs and only the successful runs, 'Succ. runs'. If the algorithm finds the optimal solution (or near optimal according to an error tolerance) to an instance in a run, then the run is considered to be a successful one. Further, 'ABT' is the average best time to find the optimal value, i.e., is the average of the best time, from the 30 runs, among all instances of a set. 'Nopt' is the number of instances that were solved to optimality (at least in one run among the 30 runs) in a set, and 'ASR' is the average, among all instances in the set, of the success rate (in %). The success rate, 'SR', indicates the percentage of the 30 runs that found the known optimal solution according to the given error tolerance.

We note that CPLEX MIP and GA (in Table 2) solve all the instances to optimality in each set.

When comparing bAFSA with IbAFSA, we may conclude that IbAFSA gives better results than bAFSA, as far as 'AIT', 'AT' and 'ASR' for the 30 runs are concerned. See Table 2. The overall average success rate of IbAFSA is 92.97%, whereas that of bAFSA is 68.12%. This is obtained mainly due to the usage of add_item and reinitialization techniques in IbAFSA. Although 'AIT' and 'AT', for the 'Succ. runs', are in general smaller in bAFSA than in IbAFSA, those values correspond to a few number of successful runs (see column 'ASR'). We note that

Table 2: Results obtained by CPLEX MIP, GA, bAFSA, GADS and IbAFSA

	CPLEX MIP		GA		bAFSA							GADS						IbAFSA						
Prob.	1 run		1 run		30 runs				Succ. runs			30 runs				Succ. runs		30 runs				Succ. runs		
set	AIT	AT	AT†	ABT†	AIT	AT	Nopt	ASR	AIT	AT	ABT	AIT	AT	Nopt	ASR	AIT	AT	AIT	AT	Nopt	ASR	AIT	AT	ABT
HP	32	0.05	2.6	0.4	737	2.10	2	28.33	74	0.26	0.09	399	0.22	2	76.67	235	0.13	189	0.40	2	98.33	176	0.37	0.06
PB	526	0.13	5.2	0.1	452	1.35	6	60.56	96	0.31	0.08	352	0.25	6	78.33	183	0.13	77	0.17	6	100.00	77	0.17	0.02
PT	22	0.06	3.2	0.2	324	1.19	7	69.52	56	0.25	0.14	335	0.24	5	71.43	70	0.04	262	0.83	7	76.19	123	0.39	0.18
SENTO	953	0.12	11.5	0.3	1712	9.95	2	15.00	84	0.61	0.50	1959	3.03	1	6.67	1379	1.92	43	0.28	2	100.00	43	0.28	0.13
WEING	12	0.14	4.3	0.4	602	4.22	8	65.83	80	0.48	0.30	665	0.76	6	70.33	184	0.09	543	3.11	8	78.75	266	1.30	0.51
WEISH	21	0.07	6.4	0.1	468	2.84	30	76.11	67	0.47	0.34	1312	1.38	17	33.33	493	0.41	109	0.56	30	98.44	89	0.45	0.08
Average	109	0.09	5.6	0.2	522	2.90		68.12	71	0.42	0.28	979	1.04		49.15	345	0.29	188	0.91		92.97	119	0.53	0.16

† Not applicable here (due to different machine used), only to show

IbAFSA performs favorably relative to the criterion 'ABT' when compared with bAFSA. Based on 'Nopt', we may conclude that bAFSA and IbAFSA also solve all instances to optimality (at least in one run among the 30). Using reinitialization of the population, IbAFSA gives more successful runs after R iterations.

When we compare the results of IbAFSA with those of GADS in Table 2, we observe that IbAFSA has a better performance than GADS relative to the criterion 'AIT', in almost all sets. The average computational time ('AT') of IbAFSA is higher than that of GADS in some sets. This is due to the procedure that aims to identify the points inside the 'visual scope' of each individual point, at all iterations. We note that GADS did not reach the optimal solution to some instances in any of the 30 runs. The 'ASR' obtained by GADS for each set are all under 80%, while IbAFSA reaches 100% in sets PB and SENTO, and almost 99% in sets HP and WEISH.

Since GADS and IbAFSA are population-based stochastic methods, we compare them further using different performance criteria: the average of the obtained objective function values, 'z_{avg}', with the standard deviation of the function values, '*std*', and the success rate, 'SR' (in %). For a clear comparison, we plot in Figs. 1–5 the bar profiles of 'z_{avg}' obtained by GADS and IbAFSA among the 30 runs, for all the instances of the six sets. The values of '*std*' relative to 'z_{avg}' are shown over the bars. We can notice that IbAFSA outperforms GADS. The 'z_{avg}' values of GADS are in general slightly smaller than those of IbAFSA, in particular on instances 16 to 30 of the WEISH set, and the '*std*' values are larger with GADS. To emphasize

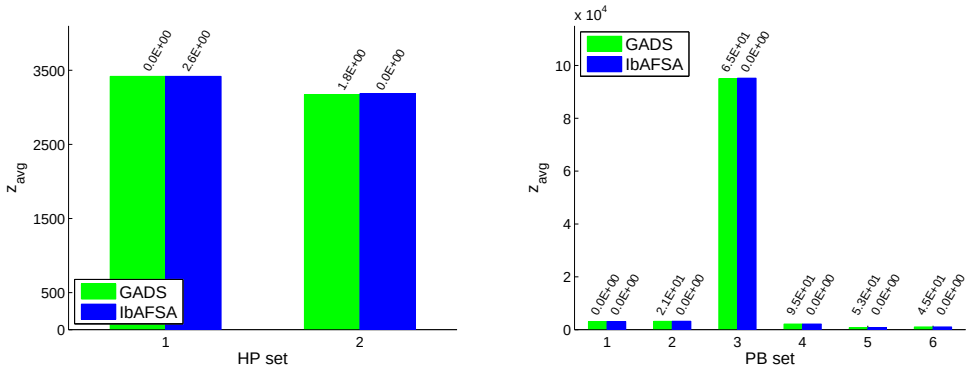

Figure 1: Comparison of z_{avg} and *std* on HP set (left) and PB set (right)

the improvement on the quality of the solutions obtained by the herein proposed IbAFSA, when compared with the preliminary version bAFSA [43] and GADS, we show in Fig. 6 the bar profiles corresponding to 'SR'. We may conclude that IbAFSA outperforms GADS and bAFSA in criterion 'SR'.

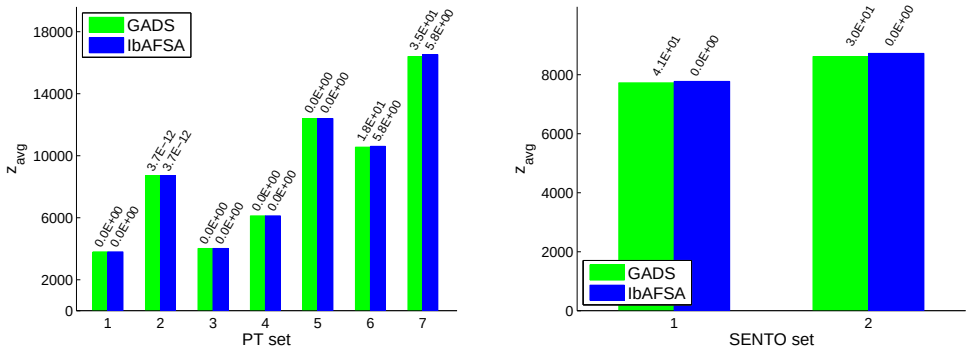

Figure 2: Comparison of z_{avg} and *std* on PT set (left) and SENTO set (right)

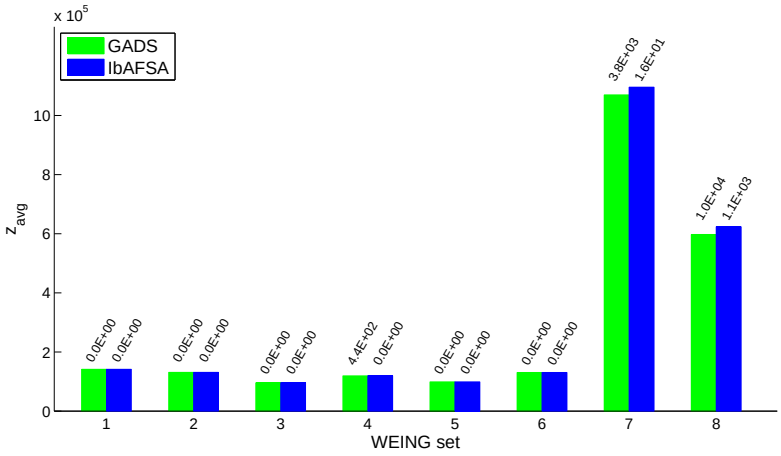

Figure 3: Comparison of z_{avg} and *std* on WEING set

In the previous section, the time complexity of one iteration of IbAFSA has been analyzed. Although the running time, per iteration, of IbAFSA could be greater than that of GADS, which is of order of complexity $\mathcal{O}(mn)$, the number of iterations required to reach the optimal (or near optimal) solution is much smaller with IbAFSA – an average over the 55 instances of 119 in IbAFSA against 345 in GADS – and it reaches the optimal solution more often – an average of 188 iterations with 92.97% of successful cases in IbAFSA against 979 iterations with 49.15% of successes in GADS, over the 30 runs (see Table 2).

We now compare IbAFSA with HGA (Hybrid Genetic Algorithm) described in [20]. For a fair comparison we run IbAFSA with $N = 5n$ and $T_{\max} = 3000$

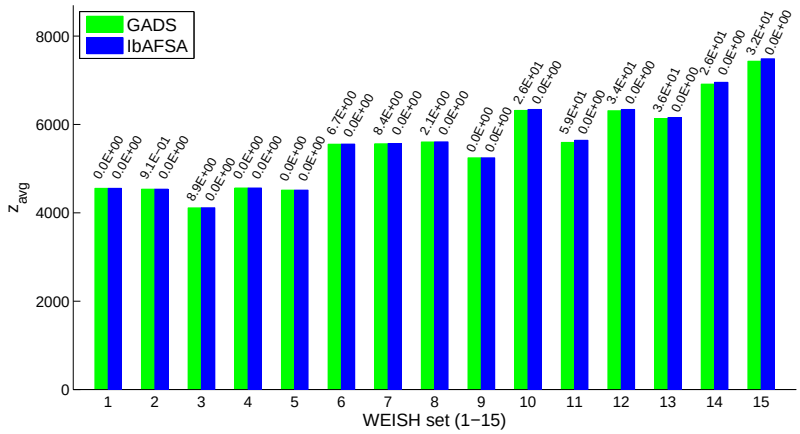

Figure 4: Comparison of z_{avg} and *std* on instances 1 to 15 from WEISH set

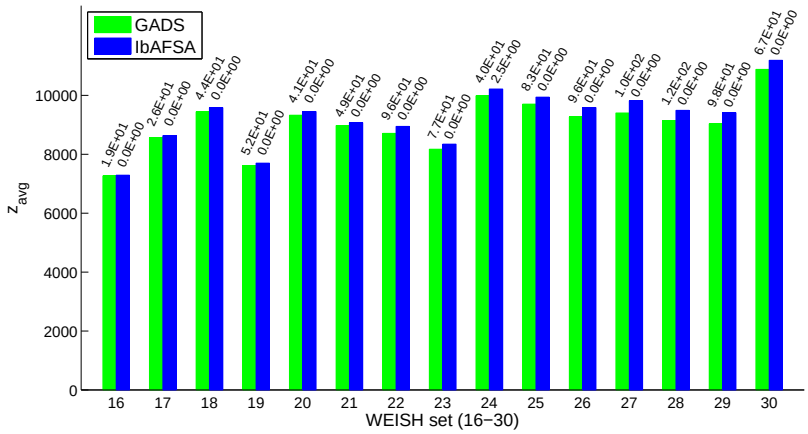

Figure 5: Comparison of z_{avg} and *std* on instances 16 to 30 from WEISH set

for all the 55 instances and 10 independent runs were carried out for each instance (same as HGA). Table 3 shows the comparison based on the different performance criteria. The data of HGA are taken from the corresponding literature. Although the machines used to obtain the results are different, IbAFSA shows a very good performance when compared with HGA.

We also compare IbAFSA with some variants of the particle swarm optimization (PSO) algorithm. Table 4 contains numerical results obtained by binary versions of PSO from the literature. The results are taken from [38, 40]. SBPSO is a Set-Based

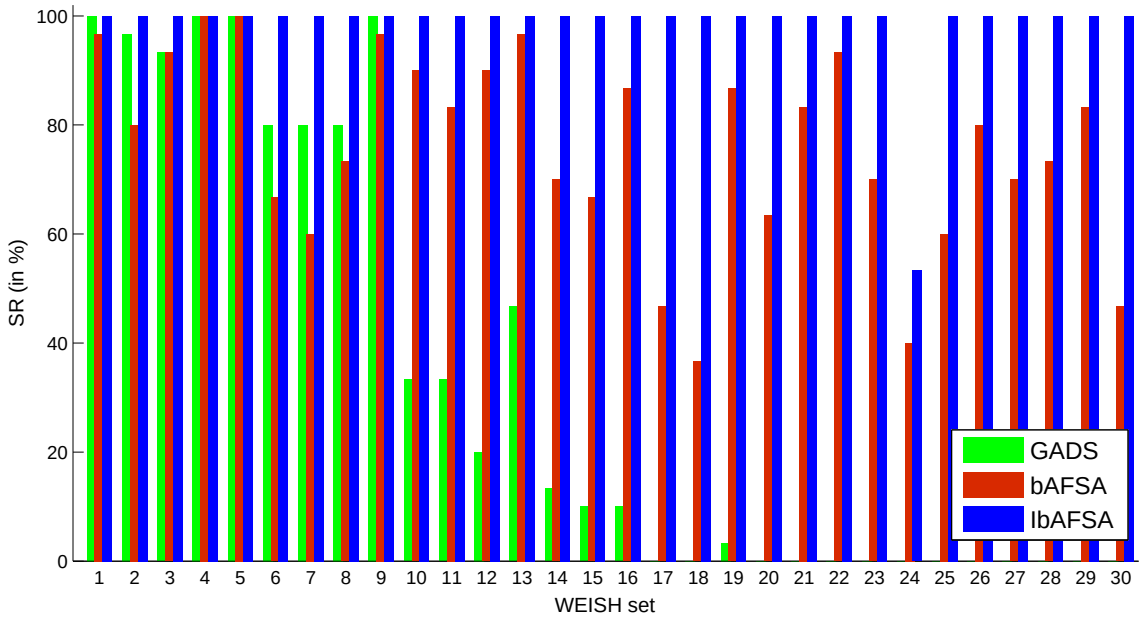

Figure 6: Comparison of SR: IbAFSA *vs.* GADS and bAFSA

PSO proposed in [40] that handles the constraints using a dead penalty. On the other hand, the modified binary PSO (MBPSO) in [38] applies a static penalty function approach to handle the constraints and uses a probability function to maintain diversity in the swarm. The results of BPSO are taken from the last cited paper. The comparisons are based on the average success rate, 'ASR', and average percentage gap to optimality, 'APG', where the percentage gap to the optimality ('PG') is defined by

$$\mathrm{PG} = \frac{z_{\mathrm{opt}} - z_{\max}}{z_{\mathrm{opt}}}\, 100\%.$$

The table shows that IbAFSA outperforms the three selected PSO versions in both criteria.

Finally, we compare the proposed IbAFSA with the heuristic methods HDP+LBC (Hybrid Dynamic Programming with Limited Branch-and-Cut) and DPHEU (Dominance Principle based Heuristic) described in [24, 27]. The results for comparison are reported in Table 5. This comparison is based on 'PG' and on 'Nopt'. We remark that the reported values for HDP+LBC and DPHEU are relative to just one run, and

Table 3: Comparison of HGA and IbAFSA

Prob.	HGA			IbAFSA		
set	ABT	AT	Nopt	ABT	AT	Nopt
HP	61.0	84.0	2	0.19	0.82	2
PB	108.0	425.0	6	0.08	0.24	6
PT	150.0	352.0	7	0.51	1.49	7
SENTO	29.0	162.0	2	0.55	0.90	2
WEING	126.0	452.0	8	4.46	26.04	8
WEISH	101.0	321.0	30	0.95	2.69	30
Average	95.8	299.3		1.27	5.53	

Table 4: Comparison of BPSO, SBPSO, MBPSO and IbAFSA

Prob.	ASR				APG			
set	SBPSO	BPSO	MBPSO	IbAFSA	SBPSO	BPSO	MBPSO	IbAFSA
HP	5.00	25.00	53.00	98.33	1.68	0.97	0.33	0.01
PB	14.50	21.50	48.17	100.00	1.81	1.35	0.56	0.00
PT	59.00	–	–	76.19	0.32	–	–	0.03
SENTO	13.50	27.00	48.00	100.00	0.45	0.22	0.10	0.00
WEING	47.50	57.50	78.13	78.75	2.49	0.34	0.11	0.02
WEISH	32.23	56.30	74.20	98.44	0.66	0.80	0.53	0.00
Average	34.25	49.63	69.63	92.97	1.04	0.78	0.44	0.01

– Not considered

the reported IbAFSA values correspond to the percentage gap to optimality of the best run ('PG best') and the average percentage gap, over the 30 runs, 'APG'. We note that these values are averaged over all the instances in a set. IbAFSA solves all the instances in the sets HP, PB, PT, SENTO, WEING and WEISH to optimality in at least one of the 30 runs. From the table, we conclude that IbAFSA gives good performance when compared with HDP+LBC and DPHEU.

Although other solution methods are available, they are not considered during the numerical comparisons with IbAFSA since they use just a few instances of each set. Based on the numerical experiments carried out until now, we may conclude that the proposed improved binary version of the artificial fish swarm algorithm is rather effective and competitive when solving the 0–1 MKP.

Table 5: Comparison of HDP+LBC, DPHEU and IbAFSA

Prob.	HDP+LBC		DPHEU		IbAFSA		
set	PG	Nopt	PG	Nopt	PG best	APG	Nopt
HP	0.45	–	0.00	2	0.00	0.01	2
PB	0.20	–	0.04	5	0.00	0.00	6
PT	0.02	–	0.00	7	0.00	0.03	7
SENTO	0.00	–	0.00	2	0.00	0.00	2
WEING	0.00	–	0.00	8	0.00	0.02	8
WEISH	–	–	0.03	28	0.00	0.00	30
Average	0.09		0.02		0.00	0.01	

– Not considered/not available in the paper

5. Conclusions

In this paper, an improved binary version of the artificial fish swarm algorithm for solving 0–1 MKP has been presented. In this method a point in the population is represented by a binary string of 0/1 bits. The Hamming distance is computed in order to identify the neighborhood points inside the 'visual scope' of a current point. Depending on the number of points inside the 'visual scope', the current point can perform either chasing, swarming, searching or random behavior. To create a trial point, each bit is generated by copying the corresponding bit from the current point or from some other specified point, with equal probability. When a leaping behavior is to be performed, some randomly chosen bits of a selected point are changed from 0 to 1, or 1 to 0, with an user defined probability. The decoding algorithm, combined also with an add_item algorithm to each feasible point, are also implemented in order to make the points feasible and improve the quality of the solution. A greedy selection criterion is used to decide whether or not the trial points should become members of the population in the next iteration. A periodic reinitialization of the population has shown to improve the quality of the solutions and increase IbAFSA consistency.

A comparison of IbAFSA with other solution methods available in the literature has been carried out with the 55 benchmark test instances. The effectiveness of IbAFSA has been shown when compared with a previous binary version of AFSA. The occurrence of obtaining the optimal solution has increased in average from 68.12% up to 92.97%. The running time of one iteration of IbAFSA for solving 0–1 MKP with m constraints and n decision variables has been analyzed and the complexity is of $\mathcal{O}(n^2)$ for fixed m, $\mathcal{O}(m)$ for fixed n and $\mathcal{O}(mn+n^2)$ for variable m and n. The comparison with GADS and HGA became highly favorable to IbAFSA.

The results show that IbAFSA outperforms GADS in all tested criteria ('AIT', 'AT', 'Nopt' and 'ASR') and HGA in the criteria 'ABT' and 'AT'. The numerical results demonstrate the superiority of the proposed IbAFSA against swarm-based heuristics such as some binary versions of the PSO algorithm, BPSO, MBPSO and SBPSO. From the comparison with the heuristics HDP+LBC and DPHEU, we conclude that IbAFSA has a competitive performance. Thus, it is found that the proposed method is rather effective and competitive when solving small benchmark instances. Future developments will focus on the large instances of the 0–1 MKP and on the simplification of some procedures, in particular those related with generating a trial point from the current one based on fish behavior, in order to reduce the processing time.

Acknowledgments

The authors would like to thank four anonymous referees for their careful reading of the paper and for their helpful comments and suggestions which greatly improved the paper. The first author acknowledges Ciência 2007 of FCT (Foundation for Science and Technology), Portugal for the fellowship grant: C2007-UMINHO-ALGORITMI-04. Financial support from FEDER COMPETE (Operational Programme Thematic Factors of Competitiveness) and FCT under project FCOMP-01-0124-FEDER-022674 is also acknowledged.

References

[1] Petersen CC. Computational experience with variants of the balas algorithm applied to the selection of R&D projects. Management Science 13(9) (1967) 736–750.

[2] Weingartner HM. Mathematical Programming and the Analysis of Capital Budgeting Problems Prentice-Hall, Englewoods Cliffs, 1963.

[3] Gavish B, Pirkul H. Efficient algorithms for solving multiconstraint zero–one knapsack problems to optimality. Mathematical Programming 31 (1985) 78–105.

[4] Shih W. A branch and bound method for the multiconstraint zero–one knapsack problem. Journal of the Operational Research Society 30 (1979) 369–378.

[5] Pisinger D. Algorithms for knapsack problems. Ph.D. thesis, Department of Computer Science, University of Copenhagen, Denmark, 1995. `http://www.diku.dk/hjemmesider/ansatte/pisinger/`

[6] Balev S, Yanev N, Fréville A, Andonov R. A dynamic programming based reduction procedure for the multidimensional 0–1 knapsack problem. European Journal of Operational Research 166 (2008) 63–76.

[7] Weingartner HM, Ness DN. Methods for the solution of the multidimensional 0/1 knapsack problem. Operations Research 15 (1967) 83–103.

[8] Fréville A, Plateau G. The 0–1 bidimensional knapsack problem: Towards an efficient high-level primitive tool. Journal of Heuristics 2 (1996) 147–167.

[9] Cabot AV. An enumeration algorithm for knapsack problems. Operations Research 18 (1970) 306–311.

[10] Schilling KE. The growth of m–constraint random knapsacks. European Journal of Operational Research 46 (1990) 109–112.

[11] Fontanari JF. A statistical analysis of the knapsack problem. Journal of Physics A: Mathematical and General 28 (1995) 4751–4759.

[12] Soyster AL, Lev B, Slivka W. Zero–one programming with many variables and few constraints. European Journal of Operational Research 2 (1978) 195–201.

[13] Lin FT. On the generalized fuzzy multiconstraint 0–1 knapsack problem. in: Proceedings of the 2006 IEEE International Conference on Fuzzy Systems. pp. 984–989, 2006.

[14] Puchinger J, Raidl GR, Pferschy U. The multidimensional knapsack problem: structure and algorithms. INFORMS Journal of Computing 22(2) (2010) 250–265.

[15] Drexl A. A simulated annealing approach to the multiconstraint zero–one knapsack problem. Computing 40 (1988) 1–8.

[16] Hanafi S, Fréville A. An efficient tabu search approach for the 0–1 multidimensional knapsack problem. European Journal of Operational Research 106 (1998) 659–675.

[17] Vasquez M, Vimont Y. Improved results on the 0–1 multidimensional knapsack problem. European Journal of Operational Research 165 (2005) 70–81.

[18] Chu PC, Beasley JE. A genetic algorithm for the multidimensional knapsack problem. Journal of Heuristics 4 (1998) 63–86.

[19] Sakawa M, Kato K. Genetic algorithms with double strings for 0–1 programming problems. European Journal of Operational Research 144 (2003) 581–597.

[20] Djannaty F, Doostdar S. A hybrid genetic algorithm for the multidimensional knapsack problem. International Journal of Contemporary Mathematical Sciences 3(9) (2008) 443–456.

[21] Kong M, Tian P, Kao Y. A new ant colony optimization algorithm for the multidimensional knapsack problem. Computers & Operations Research 35 (2008) 2672–2683.

[22] Zou D, Gao L, Li S, Wu Z. Solving 0–1 knapsack problem by a novel global harmony search algorithm. Applied Soft Computing 11 (2011) 1556–1564.

[23] Akçay Y, Li H, Xu SH. Greedy algorithm for the general multidimensional knapsack problem. Annals of Operations Research 150 (2007) 17–29.

[24] Boyer V, Elkihel M, Baz DE. Heuristics for the 0–1 multidimensional knapsack problem. European Journal of Operational Research 199 (2009) 658–664.

[25] Hill RR, Cho YK, Moore JT. Problem reduction heuristic for the 0–1 multidimensional knapsack problem. Computers & Operations Research 39 (2012) 19–26.

[26] Pirkul H. A heuristic solution procedure for the multiconstraint zero-one knapsack problem. Naval Research Logistics 34 (1987) 161–172.

[27] Veni KK, Balachandar SR. A new heuristic approach for large size zero–one multi knapsack problem using intercept matrix. International Journal of Computational and Mathematical Sciences 4(5) (2010) 259–263.

[28] Fréville A. The multidimensional 0–1 knapsack problem: An overview. European Journal of Operational Research 155 (2004) 1–21.

[29] Jiang M, Mastorakis N, Yuan D, Lagunas MA. Image segmentation with improved artificial fish swarm algorithm. in: Mastorakis N et al. (eds.), ECC 2008, LNEE, Vol. 28. Springer-Verlag, Heidelberg, pp. 133–138, 2009.

[30] Jiang M, Wang Y, Pfletschinger S, Lagunas MA, Yuan D. Optimal multiuser detection with artificial fish swarm algorithm. in: Huang DS et al. (eds.), ICIC 2007, CCIS, Vol. 2. Springer-Verlag, Heidelberg, pp. 1084–1093, 2007.

[31] Wang CR, Zhou C-L, Ma J-W. An improved artificial fish swarm algorithm and its application in feed-forward neural networks. in: Proceedings of the Fourth International Conference on Machine Learning and Cybernetics. pp. 2890–2894, 2005.

[32] Wang X, Gao N, Cai S, Huang M. An artificial fish swarm algorithm based and abc supported QoS unicast routing scheme in NGI. in: Min G et al. (eds.), ISPA 2006, LNCS, Vol. 4331. Springer-Verlag, Heidelberg, pp. 205–214, 2006.

[33] Rocha AMAC, Fernandes EMGP, Martins TFMC. Novel fish swarm heuristics for bound constrained global optimization problems. in: Murgante B et al. (eds.), Computational Science and Its Applications, ICCSA 2011, Part III, LNCS, Vol. 6784. Springer-Verlag, Heidelberg, pp. 185–199, 2011.

[34] Rocha AMAC, Martins TFMC, Fernandes EMGP. An augmented Lagrangian fish swarm based method for global optimization. Journal of Computational and Applied Mathematics 235 (2011) 4611–4620.

[35] Neshat M, Sepidnam G, Sargolzaei M, Toosi AN. Artificial fish swarm algorithm: a survey of the state-of-the-art, hybridization, combinatorial and indicative applications. Artificial Intelligence Review (2012). DOI:10.1007/s10462-012-9342-2

[36] Mirjalili S, Lewis A. S-shaped versus V-shaped transfer functions for binary particle swarm optimization. Swarm & Evolutionary Computation 9 (2013) 1–14.

[37] Suresh K, Kumarappan N. Hybrid improved binary particle swarm optimization approach for generation maintenance scheduling problem. Swarm & Evolutionary Computation 9 (2013) 69–89.

[38] Bansal JC, Deep K. A modified binary particle swarm optimization for knapsack problems. Applied Mathematics and Computation 218(22) (2012) 11042–11061.

[39] Beheshti Z, Shamsuddin SM, Yuhaniz SS. Binary accelerated particle swarm algorithm (BAPSA) for discrete optimization problems. Journal of Global Optimization (2013). DOI 10.1007/s10898-012-0006-1

[40] Langeveld J, Engelbrecht AP. A generic set-based particle swarm optimization algorithm. in: Proceedings of the 2011 International Conference on Swarm Intelligence, France 2011, pp. id-1–id-10.

[41] Langeveld J, Engelbrecht AP. Set-based particle swarm optimization applied to the multidimensional knapsack problem. Swarm Intelligence 6 (2012) 297–342.

[42] Wang L, Wang X, Fu J, Zhen L. A novel probability binary particle swarm optimization algorithm and its application. Journal of Software 3(9) (2008) 28–35.

[43] Azad MAK, Rocha AMAC, Fernandes EMGP. Solving multidimensional 0-1 knapsack problem with an artificial fish swarm algorithm. in: Murgante, B. et al. (eds.), Computational Science and Its Applications, ICCSA 2012, Part III, LNCS, Vol. 7335. Springer-Verlag, Heidelberg, pp. 72–86, 2012.

[44] Moraglio A, Johnson CG. Geometric generalization of the Nelder-Mead algorithm. in: Cowling P, Merz P (eds.), EvoCOP 2010, LNCS, Vol. 6022. Springer-Verlag, Heidelberg, pp. 190–201, 2010.

[45] Goldberg DE. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, 1989.

[46] Michalewicz Z. Genetic Algorithms+Data Structures=Evolution Programs. Springer, Berlin, 1996.